Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Manca Bizjak

Izdelava Kinect aplikacije za pomoč pri učenju fizike

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Saša Divjak

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Cilj diplomskega dela je razvoj Kinect aplikacije, ki bi uporabnikom preko naravnega uporabniškega vmesnika ponudila alternativo klasičnemu reševanju fizikalnih problemov s področja sil, energije, gibanja in Newtonove mehanike. Aplikacija naj bi bila zasnovana kot učni pripomoček pri pouku fizike v osnovnih ali srednjih šolah in naj s 3D vizualizacijo ter realnočasovnimi simulacijami fizikalnih interakcij med objekti na interaktiven način predstavi obnašanje fizikalnih sistemov. Omogoča naj interakcijo s 3D digitalnimi objekti na način, ki je podoben interakciji z objekti resničnega sveta, in manipulacijo ter spremljanje bistvenih parametrov, ki narekujejo obnašanje teles. Aplikacijo naj bi bilo mogoče v celoti krmiliti z ročnimi kretnjami, brez uporabe miške in tipkovnice.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Manca Bizjak, z vpisno številko **63110214**, sem avtor diplomskega dela z naslovom:

Izdelava Kinect aplikacije za pomoč pri učenju fizike

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saše Divjaka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Saši Divjaku za usmerjanje, pomoč in spodbudo pri izdelavi diplomskega dela. Prav tako se zahvaljujem svoji družini za vso podporo tekom celotnega študija. Zahvaljujem se tudi Jaki in vsem svojim študijskim prijateljem, ki so me spodbujali in mi stali ob strani.

Družini in vsem, ki so me podpirali in spodbujali pri študiju.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Tehnologije in orodja **3**

2.1 Microsoft Kinect 3

2.2 Unity3D 5

2.3 Povezava Kinecta in Unity3D 8

2.4 Blender 9

3 Razvoj aplikacije **11**

3.1 Prvotna zasnova 11

3.2 Edinec 12

3.3 Shranjevanje in nalaganje scen 13

3.4 Serializacija in deserializacija objektov 13

3.5 Objava aplikacije 15

4 Naravna interakcija **17**

4.1 Metanje žarkov 18

4.2 Podprte kretnje 20

4.3 Uporabniške kontrole 21

4.4 Interakcija s 3D objekti 25

5 Fizikalni vidik **29**

5.1 Fizikalno obnašanje objektov 29

5.2 Sistemi vektorjev 33

5.3 Simulacija 36

6 Primeri uporabe aplikacije **39**

7 Sklep **45**

Slike

2.1 Zgradba Kinecta 4

4.1 Globinska slika uporabnika, kot ga vidi Kinect 18
4.2 Drsnik, ki ga krmilimo preko NUI 23
4.3 Vstopni meni aplikacije 24
4.4 Glavni meni aplikacije 25

5.1 Vizualizacija vektorjev sile teže in podlage 35
5.2 Vizualizacija vektorjev sil na klancu 36
5.3 Upodobitev oblektov aplikacije z različnima senčilnikoma . . . 37

6.1 Meni za dodajanje novega fizikalnega objekta 40
6.2 Meni za osnovno manipulacijo fizikalnega objekta 41
6.3 Lebdenje nad fizikalnim objektom in prikaz povratnih informacij 41
6.4 Meniji z drsniki za nastavljanje fizikalnih parametrov 43
6.5 Rešitve fizikalnih problemov, kot jih izpiše aplikacija 44

Seznam uporabljenih kratic

FPS	frames per second
GUI	graphical user interface
IDE	integrated development environment
IR	infrared
NUI	natural user interface
RGB	red, green, blue
RGBA	red, green, blue, alpha
SDK	software development kit
USB	universal serial bus
VGA	video graphics array
WPF	Windows Presentation Foundation
XML	extensible markup language

Povzetek

Tehnološki napredek na področju vhodnih naprav za zaznavanje gibanja je omogočil razvoj številnih aplikacij z naravnim uporabniškim vmesnikom, ki se v praksi uveljavljajo na najrazličnejših področjih. Cilj diplomskega dela je razvoj Kinect aplikacije, ki bi uporabnikom preko naravnega uporabniškega vmesnika ponudila alternativo klasičnemu reševanju fizikalnih problemov s področja sil, energije, gibanja in Newtonove mehanike. Naša aplikacija bi lahko služila kot učni pripomoček pri pouku fizike v osnovnih ali srednjih šolah, saj s 3D vizualizacijo ter realnočasovnimi simulacijami fizikalnih interakcij med objekti na interaktiven način predstavi obnašanje fizikalnih sistemov. Omogoča interakcijo s 3D digitalnimi objekti na način, ki je podoben interakciji z objekti resničnega sveta in manipulacijo ter spremljanje bistvenih parametrov, ki narekujejo obnašanje teles. Aplikacijo je mogoče v celoti krmiliti z ročnimi kretnjami, brez uporabe miške in tipkovnice.

Ključne besede: Kinect, Unity3D, zaznavanje gibanja, naravni uporabniški vmesnik.

Abstract

Technological advances in the development of motion sensing input devices have enabled utilization of natural user interfaces by numerous applications that are today being put into practice in various fields. The aim of the thesis is development of such a Kinect application that would offer users an alternative approach to solving physical problems in the areas of forces, energy, movement and mechanics, with the use of a natural user interface. Our application could be utilized as a learning tool for elementary school physics, since it offers 3D visualisation and real-time physical simulations of interactions between objects and interactively demonstrates the behaviour of physical systems. It allows users to interact with digital objects in a way, similar to how we interact with real-world objects. Application allows user to manipulate and monitor significant physical parameters. Our application can be controlled with the use of hand gestures, without utilizing a mouse or a keyboard.

Keywords: Kinect, Unity3D, motion sensing, natural user interface.

Poglavje 1

Uvod

Zadnjih nekaj let smo priča razvoju vhodnih naprav, ki so sposobne zaznavati gibanje ter interpretirati uporabnikove gibe in kretnje. Med najopaznejše tovrstne naprave trenutno sodijo Leap Motion, CamBoard Pico, SoftKinetic in Microsoftov Kinect. Kljub številnim tehnološkim razlikam in razlikam v tehničnih specifikacijah med omenjenimi napravami pa vse stremijo k istemu namenu: ponuditi alternativo standardnemu načinu za komunikacijo z računalnikom preko naravnega uporabniškega vmesnika. Že več desetletij namreč miška in tipkovnica predstavljata standardni vhodni napravi za interakcijo človek-računalnik.

Vsaka moderno opremljena osnovnošolska ali srednješolska učilnica ima danes računalnik, ki lahko učiteljem služi kot učni pripomoček. Redkeje pa v učilnicah srečamo novejše tehnologije, s katerimi bi si učitelji pomagali pri učnem procesu. Uporabnost inovativne tehnologije namreč ni vedno odraz njene tehnične dovršenosti, temveč je pogosto pogojena z naborom njej prilagojenih aplikacij. Cilj diplomske naloge je razviti prav tako aplikacijo, ki bi lahko služila kot interaktivni pripomoček pri poučevanju izbranih tematik iz poglavij fizike v osnovnih in srednjih šolah. Posebnost aplikacije je v tem, da za njeno uporabo nista potrebni miška in tipkovnica, saj ta aplikacija izpostavlja naravni uporabniški vmesnik z uporabo senzorja Kinect. Na

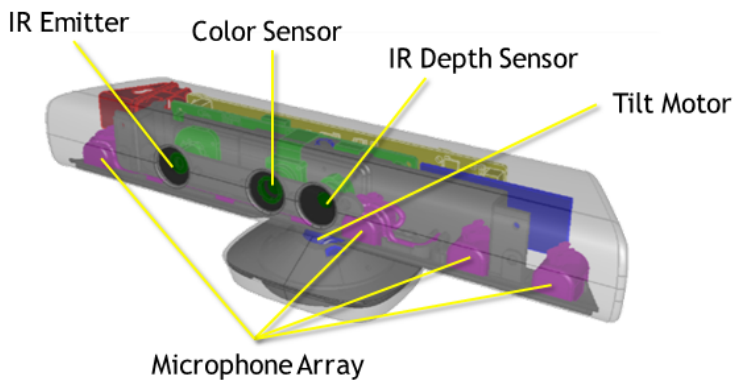
ta način se lahko uporabnik po aplikaciji navigira s pomočjo zamahov roke in lebdenjem nad gumbi, poleg tega pa sam ustvarja želeni fizikalni sistem. Z objekti digitalnega fizikalnega sistema je uporabnik lahko v interakciji na ta način, da jih premika s kretnjami, ki spominjajo na način interakcije z objekti v resničnem svetu. Aplikacija je idejno zasnovana po vzoru nekaterih tipov nalog iz fizikalnih učbenikov in predstavlja interaktivno alternativo reševanju nalog s papirjem in svinčnikom. Pri reševanju fizikalnih problemov namreč za boljšo predstavo pogosto rišemo skice, ki nam pomagajo ilustrirati stanje fizikalnega sistema v danem trenutku. Prav fizikalne simulacije v realnem času in 3D vizualizacije zato lahko služijo kot pripomoček, ki dopolnjuje klasično reševanje nalog. Razvita aplikacija bi bila lahko primerna za individualno uporabo in v učilnicah pri pouku fizike.

Poglavje 2

Tehnologije in orodja

2.1 Microsoft Kinect

Kinect je 3D globinski senzor podjetja Microsoft. Nameščen je na vrtljiv podstavek. Kinect omogoča hkraten zajem barvne in globinske slike ter zvoka. V ta namen je sestavljen iz treh ključnih komponent, in sicer iz kamere RGB, infrardečega oddajnika z globinskim senzorjem in polja štirih mikrofonov. Kamera RGB, ki podpira 8-bitno resolucijo VGA, omogoča zajem barvne slike. Za zajem globinske slike sta v Kinect vgrajena oddajnik IR in globinski senzor. Pomemben vidik naprav za zaznavanje gibanja so okvirji in njihova frekvenca, ki se meri v okvirjih na sekundo (angl. *FPS*). Okvir je paket za prenos digitalnih podatkov in si ga lahko predstavljamo kot posamezno sliko. Naprave za zajem, obdelavo in prikaz slik gibajočo sliko prikažejo kot zaporedje okvirjev. Kinect omogoča zapisovanje tako globinskih kot tudi barvnih slik s frekvenco največ 30 okvirjev na sekundo. Mikrofoni, porazdeljeni na različna mesta v podolgovati napravi, omogočajo zajem zvoka. S pomočjo podatkov iz mikrofonov je na podlagi njihove razporeditve mogoče določiti smer, iz katere prihaja posneti zvok. Vse naštete tehnične specifikacije omogočajo Kinect tehnologiji prepoznavanje kretenj in zvoka ter zajem gibanja [1]. Izgled naprave in razporeditev omenjenih komponent prikazuje slika 2.1.



Slika 2.1: Zgradba Kinecta.

Kinect kot vhodno enoto povežemo z računalnikom preko vmesnika USB, nato pa lahko podatke, ki jih zajame iz okolja, izpostavimo za uporabo najrazličnejši programski opremi. Zajete podatke Kinect pošilja računalniku v realnem času. Ti se lahko obdelajo takoj ali pa počakajo na kasnejšo analizo. V prvi vrsti je Kinect namenjen priključitvi na računalnik z operacijskim sistemom Windows in igralnima konzolama Xbox 360 ali Xbox One. Njegova ideja je, da uporabnikom omogoča način za interakcijo z računalnikom ali igralno konzolo, ki ne zahteva uporabe krmilne naprave kot je miška, tipkovnica ali igralna palica. Takemu načinu interakcije pravimo naravni uporabniški vmesnik. V tem primeru z računalnikom ali konzolo komuniciramo preko telesnih gibov oziroma kretenj ali glasovnih ukazov [2].

2.1.1 Kinect aplikacije

Področij uporabe aplikacij, ki izkoriščajo zmogljivosti Kinecta, je veliko. Obstajajo Kinect aplikacije za 3D digitalizacijo in rekonstrukcijo okolice, aplikacije z virtualno resničnostjo in prepoznavanje objektov, znani pa so tudi primeri uporabe Kinecta v robotiki, zdravstvu in izobraževanju. Uporaba Microsoftovega senzorja Kinect se je že uveljavila tako v tehnologiji iger kot tudi v akademskih okoljih. Kinect lahko uporabimo za zajem informacij

o gibanju resničnih oseb in jih nato transformiramo v digitalne, tridimenzionalne avatarje, saj nam senzor zagotavlja tridimenzionalne informacije o telesu uporabnika, prepoznavanju okostja in položaju sklepov. Naprava sama po sebi ne omogoča zaznavanja najrazličnejših gibov, specifičnih za človeško telo, saj smo ljudje sposobni izvesti zelo veliko število različnih gibov, ki jih je težko simulirati [3].

2.1.2 Uporaba Kinecta na področju izobraževanja

Po uspehih uvedbe Kinecta kot pripomočka v kirurgiji in drugod v zdravstvu se je pojavilo vprašanje, na katerih drugih področjih ima Kinect še potencial. Ena izmed atraktivnih možnosti je uporaba Kinecta kot učni pripomoček, ki bi utegnil povečati pozornost učencev in njihovo motivacijo za učenje in sodelovanje. Nedvomno se Kinect ponaša s svojo interaktivnostjo in predstavlja zanimivo alternativo klasičnemu pouku v učilnicah ali pa vsaj njegovo dopolnitev. Njegova uporaba namreč odpira kinestetični način učenja, ki za razliko od slušnega in vidnega pri pouku običajno ni prisoten. Prihodnost uporabe Kinecta na področju izobraževanja je močno odvisna od nabora temu namenjene kvalitetne programske opreme. Tudi če imajo učilnice na voljo vse potrebne kapacitete, ki so nujne za uvedbo Kinecta (računalnik, projektor, Kinect in dovolj prostora), se Kinect brez primernih izobraževalnih aplikacij nikoli ne bo uveljavil kot koristen učni pripomoček [4].

Eno izmed opaznejših gibanj, ki promovira uporabo Kinecta v učilnicah in zajema razvijalce, učitelje in učence, je KinectEDucation. Gibanje se zavzema za posodobitev in preureditev pouka, tako da bi le-ta odražal tehnološki napredek [5].

2.2 Unity3D

Razvoj in testiranje aplikacije sta v celoti potekala znotraj pogona Unity3D. Unity3D je igralni pogon z integriranim razvojnim okoljem (IDE), ki ga je

razvilo podjetje Unity Technologies. Omogoča izdelavo 2D in 3D video iger za namizne računalnike, mobilne naprave, igralne konzole in spletne brskalnike [6]. Odločitev za uporabo Unityja je temeljila na dejstvu, da ima ta že vgrajeno fizikalno okolje, poleg tega pa omogoča ustvarjanje primitivov kot so kocke, kvadri, kapsule in krogle. Kompleksnejše modele lahko ustvarimo v enem izmed programov za 3D modeliranje in jih nato uvozimo v Unity, od koder jih lahko naprej spreminjamo. Unity podpira uvoz datotek standardnih formatov iz programov kot so Maya, 3ds Max, Blender, Adobe Photoshop in drugi. Poleg 3D modelov lahko v Unity uvozimo pakete, ki vključujejo teksture, materiale, zvočne efekte, skripte, vodiče, razširitve in mnoge druge dodatke. Ti paketi so prosto dostopni ali plačljivi in jih lahko dobimo na Unityjevem spletnem portalu *Asset store*.

Unity Technologies ponuja na svojih spletnih straneh obsežno dokumentacijo in veliko število programskih vodičev, namenjenih razvijalcem z različnim predznanjem. Skupnost razvijalcev soustvarja tudi spletne strani *Unity Answers*, na katerih se v obliki odgovorov na vprašanja rešuje različne vrste bolj ali manj pogostih težav.

2.2.1 Obnašanje Unityjevih objektov

Na Unityjeve objekte lahko vplivamo tako, da jim dodamo želene komponente. Vsak objekt ima privzeto komponento *Transform*, ki omogoča nastavljanje njegovega položaja, mer in rotacije [7]. Poleg tega lahko izbiramo med številnimi komponentami kot so trkalniki, konstantne sile, spoji, toga telesa, teksture, materiali in skripte. Prav pisanje skript je obsegalo večji del razvoja, zato se jim bomo nekoliko podrobneje posvetili v nadaljevanju.

Skripte

Vsa koda aplikacije je zajeta v večjem številu Unityjevih skript. Unity omogoča programerjem pisanje skript v jezikih JavaScript, C# ali Boo. Vse skripte naše aplikacije so napisane v programskem jeziku C#. Vsaka skripta

deduje iz osnovnega razreda *MonoBehaviour* in lahko implementira funkcije, ki jih Unity samodejno kliče v predpisanem vrstnem redu [8]. Inicializacijsko kodo običajno postavimo v funkciji *Awake* in *Start*, ki se v okviru posamezne skripte izvedeta samo enkrat. Z inicializacijo vzpostavimo morebitne reference na objekte in skripte ali pa spremenljivkam nastavimo začetne vrednosti. Funkcija *Awake* je prva funkcija, ki se izvede. Če skripta vsebuje tudi funkcijo *Start*, se ta v primeru, da je instanca skripte omogočena, vedno izvede za *Awake*. To omogoča odložitev inicializacije. Omenimo še dve funkciji, ki ju prav tako uporablja veliko skript naše aplikacije. Gre za funkciji *Update* in *FixedUpdate*. Če je instanca skripte s funkcijo *Update* omogočena, se to funkcijo kliče vsak okvir. Na ta način implementiramo aplikacijsko logiko, ki se vsak okvir odziva na akcije uporabnika. Podobno velja tudi za funkcijo *FixedUpdate*, le da se ta kliče v enakomernih časovnih intervalih. Uporaba funkcije *FixedUpdate* namesto *Update* je potrebna za pravilno obnašanje togih teles.

Trkalniki

Trkalniki so koncept, ki v aplikaciji omogoča fizikalne interakcije, podobne interakcijam realnih objektov. Določajo obliko Unityjevih fizikalnih objektov in jih nanje lahko dodamo kot komponente. V aplikaciji so trkalniki objektov uporabniku nevidni, saj se največkrat uporabljajo za fizikalne interakcije. Kakšna bo interakcija med dvema objektoma s trkalnikom, je odvisno od njunih fizikalnih parametrov. Uporaba trkalnikov v aplikaciji je pomembna tudi za metanje žarkov, kot bo opisano v poglavju 4.1.

Ni nujno, da se velikost trkalnika objekta ujema z njegovo upodobitvjo. Tega se pri uporabi drsnikov poslužuje tudi naša aplikacija, kot bo opisano v poglavju 4.3. Unity ponuja 2D in 3D trkalnike. Ker so vsi objekti naše aplikacije, bodisi uporabniške kontrole bodisi fizikalni objekti, tridimenzionalni, imajo vsi 3D trkalnik. Več o teh objektih si lahko preberemo v poglavju 4. Najpreprostejši trkalniki so primitivni trkalniki, ki lahko zavzamejo obliko

krogle, kocke ali kapsule [9]. Ti so v naši aplikaciji najpogosteje uporabljani.

2.3 Povezava Kinecta in Unity3D

Za enostavno povezavo naše Unity aplikacije s Kinectom smo morali v projekt uvoziti nekaj skript, ki nam omogočajo prav to. Skripte in vzorčne primere, ki prikazujejo njihovo uporabo, smo dobili v okviru paketov v Unityjevi spletni trgovini. V naš Unity projekt smo vključili dva paketa istega avtorja - osnovnega in razširjenega. Prednost obeh paketov je, da ovijata trenutno najnovejšo različico Microsoftovega SDK-ja ter Microsoftovo izvajalno okolje. Na ta način lahko v Unity aplikacijo vključimo uporabo Kinecta in hkrati dobimo na razpolago tudi najnovejše funkcionalnosti, ki jih ponuja Microsoftov SDK.

Osnovni paket (imenovan *Kinect with MS-SDK*) je brezplačen in prikazuje uporabo mehanizma za prepoznavanje kretenj ter krmiljenje digitalnih avatarjev na podlagi uporabnikovega gibanja [10]. Razširjeni paket (*KinectExtras with MS-SDK*) se uporablja kot dodatek osnovnemu in je plačljiv, razen za uporabo v raziskovalne ali izobraževalne namene. Tako smo tudi slednjega dobili brezplačno, neposredno od avtorja. Razširjeni paket ponuja uporabo nekaterih funkcionalnosti, ki so bile v Microsoftov SDK vključene šele v različici 1.8. To so prepoznavanje zvoka, sledenje obrazu ter Kinect interakcije [11]. Izmed naštetih smo v našo aplikacijo vključili le Kinect interakcije, ki omogočajo prepoznavanje stiska ter razprtja pesti.

Ker smo v aplikaciji hkrati potrebovali določene funkcionalnoti tako prvega kot tudi drugega paketa, ju je bilo potrebno integrirati tako, da sta oba hkrati delovala znotraj istega projekta.

2.4 Blender

V aplikaciji smo potrebovali dva enostavna 3D modela teles, ki ju v Unityju ni bilo mogoče ustvariti. Za njuno izdelavo smo uporabili Blender, brezplačno odprtokodno programsko opremo za 3D računalniško grafiko, ki ima med drugim tako kot Unity integriran igralni pogon [12]. Modela, izdelana v Blenderju, smo izvozili v standardnem formatu *.obj* in ju uvozili v Unity. Čeprav je Blender orodje s široko paleto funkcionalnosti in uporabnostjo na področjih kot so animacija, modeliranje in vizualni učinki, je bil uporabljen zgolj zaradi dejstva, da Unity v prvi vrsti ni namenjen 3D modeliranju. V Blenderju smo izdelali 3D model klanca in stožcev, ki smo jih potrebovali za oblikovanje puščic vektorjev, kot je omenjeno v nadaljevanju besedila.

Poglavje 3

Razvoj aplikacije

3.1 Prvotna zasnova

Prvotna ideja in poskus implementacije aplikacije je bil drugačen od dejanske implementacije, ki je opisana v nadaljevanju. Sprva je bila predvidena neposredna uporaba uradnega Microsoftovega SDK-ja znotraj WPF projekta, ki bi ju povezali z Unityjem. V projekt so bile vključene novejše knjižnice, ki so na voljo šele od trenutno najsodobnejše verzije SDK-ja 1.8. Te med drugim omogočajo uporabo elementov uporabniškega vmesnika, ki jih je mogoče upravljati s pomočjo uporabnikovih kretenj. Na ta način nam Microsoftov SDK omogoča uporabo vnaprej pripravljenega kurzorja, ki sledi uporabnikovi roki, uporabo gumbov, ki jih sprožimo s pomočjo gest, indikatorje napredka gest za pritiske gumbov in številne druge funkcionalnosti.

Problem zgornjega pristopa se je pojavil ob integraciji WPF projekta in Microsoftovega SDK-ja z Unityjem. Čeprav povezava Unityja in WPF projekta ni težava, ki se je ne bi dalo premostiti, so se resnične težave pojavile šele zaradi dejstva, da želimo biti v realnem času v naravni interakciji z Unityjevimi objekti.

Če bi vztrajali pri odločitvi, da bomo aplikacijo izdelali kot WPF projekt

in uporabili uradni SDK, bi morali namesto uporabe Unityja oboje integrirati s knjižnico za risanje 3D objektov in objekte še povezati s fizikalnim pogonom. Poskus take integracije je bil neuspešen, zato smo se odločili, da opustimo WPF projekt in celotno aplikacijo izdelamo v Unityju. Unity ima namreč integriran lasten fizikalni pogon, prav tako pa omogoča ustvarjanje preprostih objektov, ki so v večini primerov zadostovali potrebam aplikacije.

3.2 Edinec

Edinec (angl. *singleton*) je načrtovalski vzorec, s katerim zagotovimo, da imamo vedno na voljo natanko eno instanco določenega razreda. Poleg tega je bistveno, da je do instance tega razreda omogočen globalen dostop. Za omejitev na natanko eno instanco mora poskrbeti razred sam [13]. Razredi te vrste običajno hranijo spremenljivke, do katerih pogosto dostopajo instance ostalih razredov. Uporaba edinca je v naši aplikaciji potrebna iz dveh razlogov. Prvi razlog so performančne prednosti. V Unity3D obnašanje objektov krmilimo s pomočjo skript. Če želimo iz ene skripte dostopati do javnih spremenljivk, ki so deklarirane v drugi skripti, moramo v prvi skripti pridobiti referenco na drugo. Če je druga skripta na istem objektu kot prva, to storimo s pomočjo metode *GetComponent*, vendar je to časovno potratna operacija. Časovno še bolj potraten scenarij pa je, da želimo dostopati do skripte, ki se nahaja na drugem objektu. V tem primeru moramo najprej pridobiti referenco na objekt s pomočjo metode *GameObject.Find*, nato pa še referenco na njegovo skripto. Ker je vsa aplikacijska logika zajeta v več skriptah, te pa so porazdeljene po različnih objektih, bi morali pogosto izvajati opisana scenarija. Temu se izognemo z uporabo vzorca edinec. Do edinčevih javnih spremenljivk lahko dostopajo vse skripte v aplikaciji, hkrati pa ni potrebe po ustvarjanju referenc nanj.

3.3 Shranjevanje in nalaganje scen

Ob zagonu aplikacija omogoča uporabniku izbiro fizikalnega sistema in njegovo morebitno kasnejšo zamenjavo. Ker je okolica katerega koli izmed izbranih fizikalnih sistemov v vseh primerih enaka, ni potrebe po menjavi nivojev (angl. *level*) aplikacije. Okolico sistema sestavlja ograjen prostor skupaj z vsemi kamerami aplikacije, lučmi za osvetlitev prostora in nekaterimi objekti, ki so uporabniku nevidni, vendar bistveni za delovanje aplikacije. Primer takega uporabniku skritega objekta je Unityjev prazni objekt, ki mu je dodana samo skripta, ki vsebuje edinca. Kot smo pojasnili v poglavju 3.2, je instanca tega razreda bistvena za prenos ključnih parametrov aplikacije pri menjavi fizikalnega sistema in njegovemu shranjevanju ter nalaganju. Aplikacija vsebuje še nekaj praznih objektov, ki so namenjeni semantičnemu strukturiranju in gradnji hierarhije objektov. Poleg tega so na take objekte dodane ključne vedenjske skripte, ki narekujejo celotno obnašanje aplikacije. Ker ni potrebe po menjavi oziroma uničenju in ponovnem ustvarjanju navedenih okoliških objektov, smo se odločili, da namesto sistema nivojev v aplikaciji uporabimo instanciranje in uničevanje sredstev (v Unityjevi terminologiji imenovana *prefabs*). S tem pristopom opišemo vsak fizikalni sistem z lastnim sredstvom, ki se instancira na zahtevo uporabnika. Ko uporabnik zamenja fizikalni sistem, se obstoječi uniči in ustvari na novo izbrani, okoliški objekti pa ostanejo enaki znotraj posameznega zagona aplikacije. Gre za tako imenovano obstojnost objektov.

Ob zagonu aplikacije ima uporabnik tri možnosti: izbere eno izmed vnaprej pripravljenih scen, ustvari prazno sceno, lahko pa naloži katero izmed scen, ki jih je sam predhodno shranil.

3.4 Serializacija in deserializacija objektov

Serializacija je mehanizem, s pomočjo katerega lahko podatkovne strukture ali stanje programskih objektov predstavimo v različnih formatih, npr. teks-

tovnem, XML ali binarnem. Serializaciji obraten postopek je deserializacija, ki iz podatkov v dani obliki izgradi objekte oziroma podatkovne strukture. Deserializacija se lahko izvede v istem ali drugem računalniškem okolju kot serializacija. Omenjena postopka se običajno uporabljata za prenos objektov preko omrežja ali pa njihovo shranjevanje na izbrani medij (npr. datoteko ali podatkovno bazo)[14, 15].

V aplikaciji uporabljamo serializacijo za shranjevanje stanja vseh pomembnih podatkov o stanju fizikalnega sistema in o objektih v datoteko, deserializacijo pa za obnovitev stanja sistema. Na ta način lahko uporabnik v aplikaciji spreminja obstoječ fizikalni sistem, ga shrani in nato kadarkoli ponovno odpre. Aplikacija uporablja binarno serializacijo, ki jo omogoča C#. Zaradi uporabe binarne serializacije in ne XML ali tekstovne serializacije, je uporabniku stanje serializiranih objektov prikrito, zato jih je bistveno težje načrtno spreminjati kot sicer. Serializira se en sam razred, imenovan *AppData*, ki je pravzaprav zgolj vsebovalni razred za vse podatke, ki jih želimo shraniti. Ko uporabnik zahteva shranjevanje scene, se najprej ustvari nova instanca tega razreda, nato pa se vsi njegovi atributi napolnijo z aplikacijskimi podatki. Ti podatki vključujejo število vseh teles, vrsto vsakega posameznega telesa, njihove položaje, rotacije, velikosti, mase, zračne upore, koeficiente trenja in barve. Z deserializacijo navedenih podatkov lahko v celoti rekonstruiramo stanje fizikalnega sistema. Ker shranjujemo samo enostavne podatkovne tipe (boolove spremenljivke, števila, polja, sezname) in ne objektov samih, je potrebno objekte ob nalaganju shranjene scene ponovno ustvariti in jim ustrezno nastaviti parametre.

Serializirani podatki se shranijo v datoteko s končnico *.dat*. Ime datoteke sestavljata ime shranjene scene in časovni žig. Datoteka se shrani v privzeti imenik, do katerega Unity vedno lahko dostopa. Pot do tega imenika je odvisna od operacijskega sistema, referenco nanjo pa v Unityju dobimo preko *Application.PersistentDataPath*.

3.5 Objava aplikacije

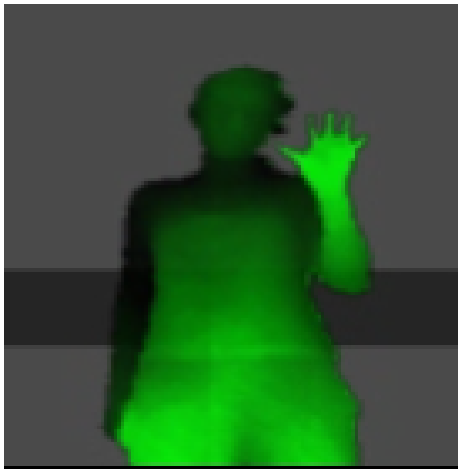
Po končanem razvoju je potrebno aplikacijo še zgraditi (angl. *build*), da jo lahko objavimo in poženemo kot samostojno aplikacijo. Aplikacija je bila izgrajena ter testirana na 32- in 64-bitnem sistemu Windows, kjer je potekal tudi njen razvoj. Rezultat izgradnje aplikacije je izvršljiva datoteka, poleg tega pa se samodejno ustvari tudi imenik, ki vsebuje vsa sredstva, potrebna za njen zagon in pravilno delovanje [16].

Poglavje 4

Naravna interakcija

Bistvo aplikacije je, da za interakcijo ne potrebujemo miške ali tipkovnice. Temu pravimo, da uporabljamo naravni uporabniški vmesnik. Celotna navigacija poteka s pomočjo ročnih kretenj, ki jih aplikacija zaznava na podlagi podatkov iz senzorja. Za zanesljivo delovanje mora uporabnik stati vsaj 1,5 metra od senzorja. Ko aplikacija zazna uporabnika, se v spodnjem desnem vogalu prikaže njegova silhueta, ki mu daje neposredno informacijo o tem, kaj senzor zaznava v vsakem trenutku. Uporabnika aplikacije, kot ga vidi senzor, prikazuje slika 4.1. Čeprav senzor med seboj lahko razloči več človeških silhuet, razvita aplikacija predvideva, da z njo komunicira en uporabnik.

Elemente aplikacije, ki jih lahko krmilimo s pomočjo rok, lahko razdelimo v dve kategoriji. Prva kategorija obsega 3D elemente, ki imajo fizikalne lastnosti in so lahko v medsebojni interakciji z ostalimi takimi objekti. V to kategorijo sodijo krogle, kocke, kvadri in klanci, ki tvorijo nek fizikalni sistem. Na te objekte lahko uporabnik s svojimi kretnjami neposredno vpliva, lahko jih zagrabi, premika, spusti, pritisne in jim spreminja velikost. Druga kategorija elementov aplikacije pa obsega komponente uporabniškega vmesnika, ki jih uporabnik vidi kot 2D objekte (v Unityju so ti implementirani kot 3D). Te komponente so uporabniški meniji, drsniki, gumbi in oznake. Na stanje drsnikov lahko uporabnik neposredno vpliva, ostale uporabniške kontrole pa



Slika 4.1: Uporabnik aplikacije, kot ga vidi Kinect.

so namenjene zgolj manipulaciji opazovanega fizikalnega sistema ali stanja aplikacije.

4.1 Metanje žarkov

Metanje žarkov (angl. *ray casting*) je ena izmed temeljnih tehnik za upodabljanje, ki se v računalniški grafiki pogosto uporablja za reševanje raznovrstnih problemov. Gre za algoritem, kjer iz točke, ki predstavlja položaj opazovalca, zgradimo navidezni svetlobni žarek v določeni smeri. V kontekstu naše aplikacije se metanje žarkov uporablja za zaznavanje objektov, ki ležijo na poti žarka. Pogosto želimo namreč ugotoviti, ali ima žarek presečišče z opazovalcu najbljižjim od upodobljenih objektov. Presečišče z objektom, ki je opazovalcu najbližje, določa, kaj bo prikazano na določeni zaslonski točki. Metanje žarkov se torej uporablja za upodabljanje 3D scen na 2D površino zaslona in predstavlja način za določanje, kaj bo prikazano na posamezni zaslonski točki. [17, 18, 19].

Metanje žarkov ima za našo aplikacijo zelo velik pomen. Lahko bi rekli, da na njem sloni celotna naravna interakcija. Omogoča nam, da s kurzor-

jem miške, ki sledi uporabnikovi dlani, izbiramo 3D objekte ali krmilimo gumbe uporabniškega vmesnika. Položaj uporabnikove dlani se preslika v položaj kurzorja na zaslonu, ta pa predstavlja izhodiščno točko, od koder ustrelimo navidezni žarek v smeri naravnost naprej glede na položaj glavne kamere. Če žarek seka gumb ali 3D objekt, potem kurzor lebdi nad njim, kar je za aplikacijo lahko indikator, da želi uporabnik nad njim izvesti neko akcijo. Algoritem metanja žarkov se v aplikaciji izvaja vsak okvir na dveh različnih plasteh. Omenili smo namreč dva primera interakcije - z gumbi ali s 3D objekti. Ker gre za semantično in funkcionalno različni skupini elementov, je smiselno, da ju obravnavamo ločeno. Pri tem smo izkoristili Unityjev sistem plasti, ki omogoča razporejanje objektov na različne plasti, ki jih lahko ustvarjamo tudi sami, in omogoča njihovo različno obravnavo. Gumbe uporabniškega vmesnika smo postavili v plast GUI - plast grafičnega uporabniškega vmesnika, 3D objekte pa smo pustili v privzeti plasti. Ker morajo biti objekti obeh plasti upodobljeni na popolnoma drugačen način (gumbi so 2D objekti, ki jih želimo prikazati nad ostalimi objekti, ki so 3D), smo uporabili dve različni kameri. Prav kamere v Unityju najpogosteje izkoriščajo sistem plasti, saj jim ta omogoča, da upodobijo samo izbrani del scene [20]. Ena izmed naših kamer tako upodablja objekte v sceni, ki se nahajajo na GUI plasti, druga pa upodablja privzeto plast. Za upodobitev GUI plasti nad privzeto plastjo smo GUI kameri nastavili večjo globino kot kameri, ki upodablja privzeto plast.

Možnost izvajanja metanja žarkov Unity programerju izpostavi preko funkcij različnih razredov. V aplikaciji uporabljamo funkcijo *Raycast* razreda *Physics*, ki išče presečišča žarka z vsemi objekti na sceni, ki imajo na sebi trkalnik. Unity programerju nudi možnost, da sam nastavi začetno točko, smer in dolžino ustreljenega navideznega žarka. Dolžina žarka je lahko končna ali pa neskončna. Poleg tega je možno preveriti samo presečišča z objekti, ki se nahajajo na določenih plasteh [19, 20]. V aplikaciji tako ločeno izvajamo metanje žarkov na GUI plasti in na privzeti plasti.

4.2 Podprte kretnje

Uporabnik v celoti krmili aplikacijo s pomočjo ročnih kretenj. V nadaljevanju podajamo seznam podprtih kretenj in opis njihove pravilne izvedbe. Uporabljena Unity paketa z vgrajenim zaznavanjem kretenj poleg navedenih kretenj podpirata še številne druge. Učinke navedenih kretenj na obnašanje aplikacije bo bralec spoznal v poglavjih 4.4 in 4.3.

- **Klik**
  Razprto dlan (levo ali desno) umirimo in vsaj 2 sekundi držimo v istem položaju.

- **Pritisk**
  Z razprto dlanjo (levo ali desno) premaknemo roko naravnost naprej proti senzorju. Gib je potrebno izvesti v času 1,5 sekunde.

- **Stisk pesti**
  Razprto dlan (levo ali desno) stisnemo v pest.

- **Sprostitev pesti**
  Razklenemo stisnjeno pest leve ali desne roke in razpremo dlan.

- **Zamah v levo**
  Z desno roko zamahnemo v levo.

- **Zamah v desno**
  Z levo roko zamahnemo v desno.

- **Zamah navzgor**
  Z levo ali desno roko zamahnemo navzgor.

- **Zamah navzdol**
  Z levo ali desno roko zamahnemo navzdol.

Iz zgornjega opisa je razvidno, da lahko z izjemo zamaha v levo in zamaha v desno vse kretnje izvajamo z levo ali desno roko.

4.3 Uporabniške kontrole

V primeru, da bi aplikacijo izdelali kot v poglavju 3 omenjeni WPF projekt in pri tem uporabili uradni Microsoftov SDK, bi imeli na razpolago nekaj elementov uporabniškega vmesnika kot so meniji, odzivni gumbi, drsniki, kurzor, ki sledi roki. Odločitev za izdelavo aplikacije v Unityju ima poleg številnih prednosti tako tudi nekaj pomanjkljivosti. Na novo je bilo potrebno oblikovati osnovne gradnike uporabniškega vmesnika in sprogramirati njihove funkcionalnosti tako, da jih je mogoče krmiliti s pomočjo Kinecta. V primeru uporabe uradnega SDK-ja bi bilo to že vgrajeno v okviru ustreznih knjižnic.

4.3.1 Gumbi

Uporabnik lahko izvede klik na gumb uporabniškega vmesnika tako, da nad njega s premikom dlani postavi kurzor miške, nato pa vsaj eno sekundo drži dlan v istem položaju. Ko aplikacija zazna klik določenega gumba, se izvede ustrezna akcija. Gumbi so namenjeni tako navigaciji po menijih aplikacije kot tudi manipulaciji fizikalnega sistema. Iz besedilne oznake nad posameznim gumbom je jasno razviden njegov namen, torej katero akcijo sproži. Ko aplikacija zazna lebdenje kurzorja nad nekim gumbom, se gumb poveča, ob umiku kurzorja pa se zopet vrne v originalno velikost. Povečanje gumba uporabniku zagotavlja povratno informacijo.

4.3.2 Drsniki

V aplikaciji na več mestih ponudimo uporabniku nastavljanje določenega parametra. Izbor vrednosti parametra bi povsod lahko implementirali s pomočjo množice večjega števila gumbov, vendar bi na ta način močno omejili uporabnikovo svobodo pri izbiranju. Na ta način je v aplikaciji realizirana izbira barve objekta ali dodajanje novega klanca s specifičnim naklonom (10, 15, 20, 25, 30 in 35 stopinj). V tem primeru je tak pristop še smiseln, saj je zaloga vrednosti relativno majhna, poleg tega pa vsebuje klance z nakloni, ki se v fizikalnih nalogah najpogosteje pojavljajo. Ta pristop ni najbolj pri-

meren za nastavljanje parametrov z večjimi zalogami vrednosti. Potrebovali bi namreč veliko število gumbov, posledično pa bi velik del zaslona zasedli z uporabniškimi kontrolami. Ta problem smo rešili z izdelavo drsnikov, ki omogočajo izbiro vrednosti iz poljubno velikih diskretnih intervalov.

Drsnik je uporabniška kontrola, s katero je uporabnik v interakciji tako, da zagrabi gibajoči del kontrole (stisne pest), ter jo premakne levo ali desno oziroma gor ali dol in s tem nastavi dani parameter na želeno vrednost. Za konec interakcije z drsnikom mora uporabnik sprostiti stisnjeno pest. Zgornja in spodnja meja intervala sta določena za vsak drsnik posebej glede na vrsto parametra, ki ga nastavlja, in sta navedeni na drsniku. Za takojšnjo povratno informacijo se med interakcijo ob vsaki spremembi nemudoma spremeni oznaka na drsniku, ki odraža trenutno vrednost parametra. V aplikaciji uporabljamo vodoravne in navpične drsnike, princip delovanja pa je pri obeh enak. Gibajoči element drsnika lahko premikamo v smeri levo ali desno oziroma navzdol ali navzgor samo med predpisano spodnjo in zgornjo mejo. Položaj gibajočega elementa prevedemo na ustrezni interval. Gibajoči element drsnika je edini objekt aplikacije, ki ima trkalnik skoraj dvakrat večji od svoje upodobitve. To je posledica praktičnih in estetskih razlogov. Trkalnik mora biti zadosti velik, da bo aplikacija zaznala stisk uporabnikove pesti. Ker pri stisku pesti uporabnik pogosto nenamerno premakne roko, lahko zgreši gibajoči element. Ob premajhnem trkalniku bi moral uporabnik večkrat poskusiti zagrabiti gibajoči element, preden bi mu uspelo. Po drugi strani pa se hočemo izogniti preveliki upodobitvi, saj je eden izmed razlogov za uporabo drsnikov tudi ta, da ne zasedajo veliko prostora. Za vsak parameter, ki ga je mogoče spreminjati z drsnikom, je predpisan korak, ki predstavlja razliko med dvema zaporednima vrednostima. Odločitev za prevedbo na diskretni interval je posledica dejstva, da so parametri v fizikalnih nalogah pogosto zaokroženi. Primer drsnika prikazuje slika 4.2.

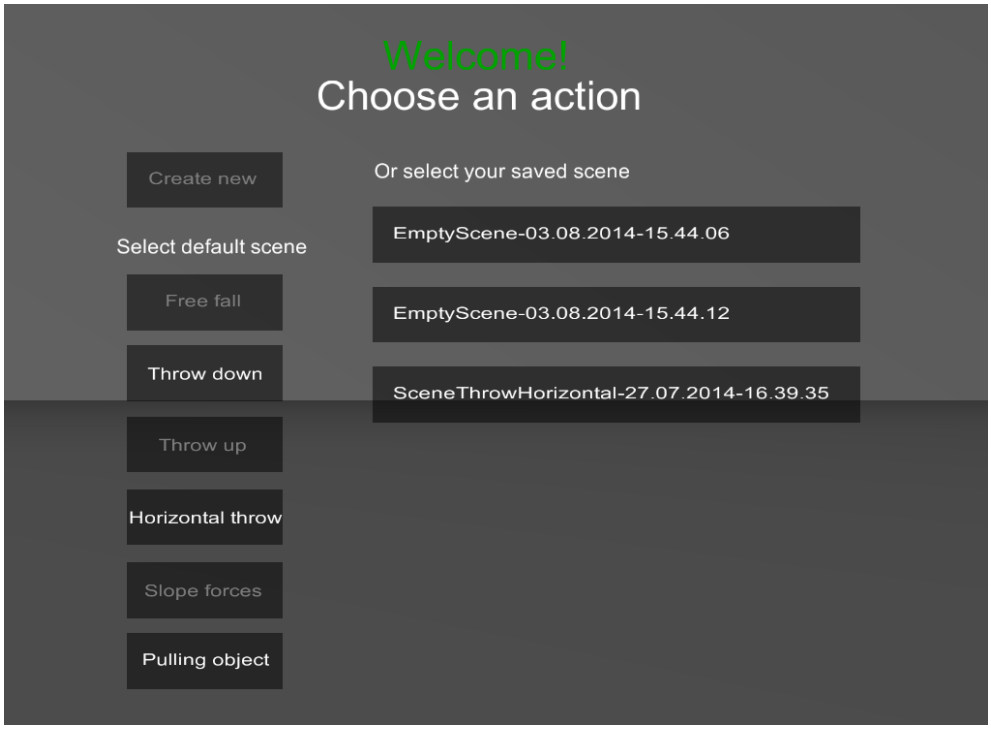
V aplikaciji se srečamo z drsnikoma za rotacijo zornega kota kamere v

Slika 4.2: Drsnik za natančnejše nastavljanje višine telesa.

vodoravni in navpični smeri, drsnikom za nastavljanje mase, zračnega upora, koeficienta trenja in višine telesa, pa tudi za izbiro začetne hitrosti telesa, konstantne sile, ki deluje na telo, ter kota, pod katerim deluje konstantna sila.

4.3.3 Meniji

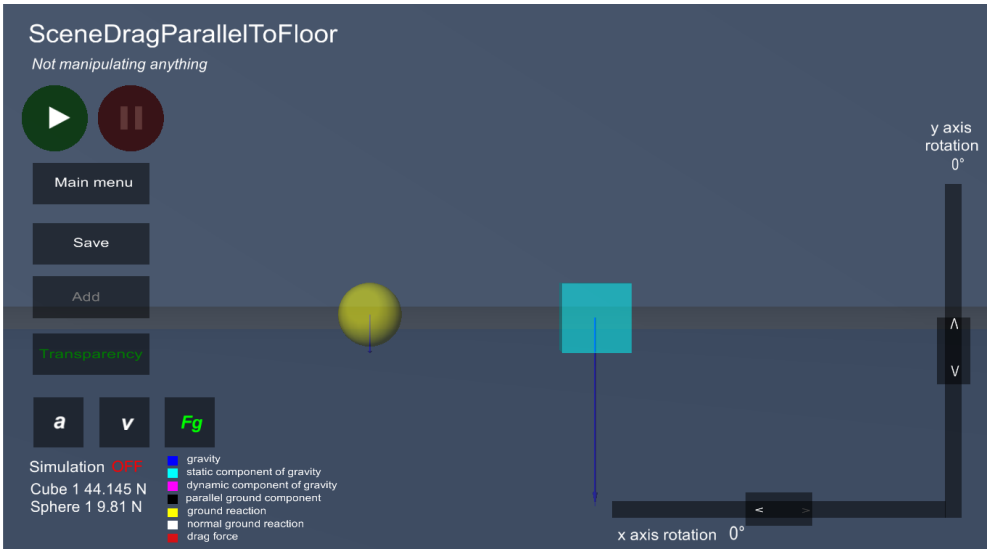
Meniji so namenjeni semantičnemu grupiranju gumbov, drsnikov in oznak. Aplikacija razlikuje med glavnim in vstopnim menijem ter t.i. meniji v ospredju. Menije v ospredju vedno odpremo s klikom na ustrezen gumb, zapremo pa z zamahom v levo. Če je v aplikaciji aktiven katerikoli meni v ospredju, se dokler je odprt, onemogoči metanje žarkov, ki ni vezano na uporabniške kontrole tega menija. Na ta način onemogočimo zaznavanje fizikalnih objektov in vseh gumbov glavnega menija. Glavni razlog za omejitev metanja žarkov je izogibanje neželeni manipulaciji fizikalnih objektov za menijem. Dokler je v aplikaciji odprt nek meni v ospredju, predvidevamo, da želimo biti v interakciji samo z njim, ne pa tudi z objekti, upodobljenimi v njegovem ozadju. Drugi razlog je optimizacija. Veliko kode v aplikaciji se izvede za vsak okvir, med drugim tudi metanje žarkov. Predvidenje kode, za katero ni nujno, da se izvede vsak okvir, lahko prihrani procesorski čas, kar se odraža tudi v tem, da aplikacija teče bolj tekoče. Ko meni v ospredju zapremo, se ta umakne iz vidnega področja kamere, ki upodablja plast uporabniških kontrol.



Slika 4.3: Vstopni meni aplikacije.

Vstopni in glavni meni

Vstopni meni vsebuje gumbe za izbiro nove, predpripravljene ali predhodno shranjene fizikalne scene. Ko izberemo eno izmed želenih scen, smo preusmerjeni na glavni meni, v katerem je mogoča interakcija z objekti fizikalnega sistema. Po preusmeritvi na glavni meni lahko do vstopnega menija pridemo s klikom na gumb *Main menu* ali z zamahom roke v levo. Iz vstopnega menija lahko ponovno pridemo do glavnega s klikom na gumb *Resume* ali z zamahom roke v desno. Pri vsakem prehodu med vstopnim in glavnim menijem se spremeni položaj kamere, ki služi za upodabljanje kontrol uporabniškega vmesnika. To se razlikuje od menijev v ospredju, saj v tem primeru spreminjamo položaj kamere, ne pa položaja samih menijev.



Slika 4.4: Glavni meni aplikacije. Na sliki je prikazan vzorčni fizikalni sistem. Prikazani so vektorji sil, za njihovo boljšo vidljivost pa je vključena prosojnost.

4.4 Interakcija s 3D objekti

4.4.1 Lebdenje nad objektom

Ko uporabnik z dlanjo premakne kurzor tako, da je ta direktno nad nekim 3D objektom v fizikalnem sistemu, se temu objektu spremeni barva. S tem zagotovimo interaktivnost in povratno informacijo, saj uporabnik vidi, da se aplikacija odziva na premike njegove roke. V eno izmed statusnih oznak se za čas lebdenja izpišejo osnovne informacije o objektu, kot so njegovo ime in fizikalne dimenzije. Lebdenje s kurzorjem nad objektom je predpogoj za nadaljnje manipuliranje objekta, npr. premikanje ali spreminjanje velikosti. Dokler uporabnik lebdi nad določenim objektom, ga lahko s pritiskom nanj izbere za morebitno nadaljnjo akcijo. Ob izvedbi kretnje za pritisk na telo (razprostrto dlan premaknemo naprej v prostoru) se namreč odpre meni, ki omogoča osnovno manipulacijo izbranega objekta.

4.4.2 Premikanje

Ko nad določenim 3D objektom v sceni lebdimo, ga lahko premaknemo na drug položaj v prostoru. Za čim bolj intuitivno premikanje objektov, ki je podobno interakciji z realnimi objekti, moramo izbrani objekt najprej zgrabiti, ga premakniti na želeno lokacijo, nato pa izpustiti. Aplikacija v ta namen opazuje, če bo uporabnik nad izbranim objektom izvedel kretnjo za stisk pesti. Ko uporabnik zgrabi objekt, lahko stisnjeno pest premika po prostoru, premiki njegove roke pa bodo odražali nov položaj 3D objekta v sceni. Premikanje se zaključi, ko aplikacija zazna kretnjo *release* (uporabnik spusti pest).

4.4.3 Spreminjanje velikosti

Po pritisku telesa lahko iz menija za manipulacijo kliknemo gumb *Resize uniform* in s tem začnemo enakomerno spreminjanje velikosti objekta (v vse smeri za enak faktor). Vse objekte lahko povečujemo enakomerno, kvadre in klance pa tudi po izbrani dimenziji. V tem primeru se v meniju prikaže tudi gumb *Resize specific*, ki omogoča spreminjanje velikosti objekta v izbrani koordinatni smeri (višino, širino ali dolžino). V obeh primerih se velikost objekta spreminja premo sorazmerno z razdaljo med obema dlanema. Ves čas nastavljanja velikosti telesa se nam v eno izmed oznak izpisujejo trenutne velikosti robov za kocke ali kvadre, naklona za klance in radija za krogle. Ko želimo prenehati s spreminjanjem velikosti objekta, moramo vsaj eno sekundo ohraniti roki v istem položaju.

Unity privzeto umerja objekte iz njihovih središč. To pri spreminjanju velikosti objektov, ki se nahajajo na neki podlagi, povzroči, da se deloma pogreznejo vanjo. To smo rešili z ustrezno spremembo položaja takih objektov po vsaki spremembi velikosti.

4.4.4 Dodajanje objekta

Na vsako izmed izbranih scen lahko poleg spreminjanja obstoječih 3D objektov dodajamo tudi nove. Do menija za dodajanje, kjer izberemo želeni tip objekta, pridemo s klikom na gumb *Add*. Uporabnik lahko doda eno ali več krogel, kock, kvadrov ali klancev. Vsak nov objekt se ustvari z določenimi privzetimi nastavitvami. Ob dodajanju klanca uporabniku ponudimo predpripravljene klance z nakloni 10, 15, 20, 25, 30 in 35 stopinj. Po izboru naklona se klanec postavi na privzeto mesto na tleh. Pri izboru katere koli druge vrste objekta pa ima uporabnik več svobode. Nov objekt namreč sledi položaju kurzorja, aplikacija pa pazi, da objekta ne moremo postaviti pod tla. Ko končamo s postavljanjem objekta, ga lahko odložimo na izbrano mesto z izvedbo kretnje *push*.

4.4.5 Druge akcije nad objekti

Po odprtju menija za manipulacijo lahko izbranemu objektu spremenimo barvo in mu z drsnikom nastavljamo maso in zračni upor. Privzeta vrednost zračnega upora telesa je nič. Če telesu povečamo zračni upor, se bo to odražalo v počasnejšem padanju telesa ob zagonu simulacije. S klikom na gumb *Remove* objekt odstranimo iz fizikalnega sistema.

Poglavje 5

Fizikalni vidik

Poleg naravne interakcije z objekti je pomemben vidik aplikacije tudi spremljanje in manipulacija njihovega fizikalnega obnašanja. Aplikacija ponuja uporabniku nastavljanje številnih začetnih parametrov, ki ob zagonu simulacije vplivajo na stanje izbranega objekta v vsakem trenutku. Eden izmed začrtanih ciljev, ki jih želi aplikacija izpolniti, je njena uporabnost pri poučevanju fizike v osnovnih in srednjih šolah. Aplikacija naj bi omogočila interaktivno reševanje fizikalnih nalog, ki ilustrirajo delovanje sil, delo in energijo ter Newtonove zakone mehanike teles.

5.1 Fizikalno obnašanje objektov

Pri fizikalnih simulacijah nam je v veliko pomoč fizikalni pogon, ki je vgrajen v Unity. Fizikalni pogon je vmesna programska oprema, ki v računalniški grafiki omogoča aproksimacijo obnašanja realnih fizikalnih sistemov v domenah dinamike in zaznavanja trkov togih teles ter dinamike tekočin in mehkih teles [21]. Fiziko v Unityju omogoča lastniški 3D pogon PhysX podjetja NVIDIA, ki ga uporabljajo številne video igre [22]. Fizikalno obnašanje objektov v naši aplikaciji smo dosegli z uporabo fizike togih teles. Vsi 3D fizikalni objekti, s katerimi je uporabnik lahko v interakciji v okviru aplikacije, so toga telesa. Če objekt v Unityju eksplicitno označimo kot tog, bo z njim

upravljal Unityjev fizikalni pogon, zato bo nanj delovala gravitacija, prav tako pa je nanj mogoče učinkovati z ostalimi silami in navorom [23]. Prav z uvajanjem delovanja nenadnih ali konstantnih sil na toga telesa v aplikaciji ob zagonu simulacije dosežemo realistično gibanje in trke med telesi.

5.1.1 Toga telesa

Naša aplikacija omogoča uporabniku interakcijo z različnimi objekti. Privzeli smo, da so klanci, stene in tla statični objekti, kocke, kvadri in krogle pa dinamični. Dinamičnim objektom lahko uporabnik izbere gibanje, ki ga želi spremljati, in glede na izbrano vrsto gibanja ustrezno spreminja parametre, ki vplivajo na potek le-tega. Vloga statičnih objektov je zgolj v tem, da nastopajo kot okolica in s tem vplivajo na obnašanje dinamičnih objektov. Odločitev, da statični objekti ne bodo imeli enake vloge kot druge vrste objektov, izhaja iz tipičnih primerov osnovnošolskih ali srednješolskih nalog iz fizike. Običajno v njih opazujemo gibanje preprostih teles v interakciji s klancem, vodoravno podlago ali steno, tako da v skoraj vseh primerih omenjena telesa nastopajo zgolj kot telesa iz okolice, ne pa kot opazovana telesa. Presodili smo, da takó opazovanje npr. vodoravnega meta klanca ne bi bilo zares smiselno. V primeru trka statičnega togega telesa z dinamičnim se bo dinamično telo odbilo in premaknilo od statičnega, ne pa tudi obratno.

5.1.2 Trki teles

Ko fizikalni pogon zazna trk dveh teles ali konec njune interakcije, lahko pokliče določene funkcije, v katerih je koda z želenim odzivom na zaznani dogodek. Te funkcije v aplikaciji uporabljamo na dveh mestih - pri prikazu vektorjev teže in podlage in pri zaznavanju ali je opazovanega gibanja telesa konec. Če je krogla na klancu, potem se bosta sila teže in podlage ustrezno razstavili na svoje komponente, skladno z naklonom klanca. Ko poženemo

simulacijo in se krogla odkotali s klanca na tla, bo aplikacija zaznala, da med kroglo in klancem ni več stika, zato opisane komponente sil ne bodo več prikazane. Aplikacija bo namesto tega zaznala trk s tlemi, zato bo prikazala silo teže in podlage telesa na vodoravni podlagi. V primeru, da se krogla odbije s tal v zrak, bo fizikalni pogon zaznal, da krogla nima stika z nobeno podlago, zato sila podlage ne bo prikazana, dokler bo telo v zraku. Drug primer uporabe funkcij za zaznavanje trkov in stikov med telesi je v primeru, da danemu telesu nastavimo neko opazovano gibanje z začetno pozicijo nad tlemi. Aplikacija sproti beleži izbrane fizikalne parametre (delne rezultate) vse dokler telo ne prileti na tla, ob trku telesa s tlemi pa izpiše končni rezultat.

5.1.3 Gibanja

Za vsak dinamičen objekt na sceni lahko uporabnik izbira med prostim padom, metom navzdol, navpičnim metom oziroma metom navzgor, vodoravnim metom in potiskanjem objekta s konstantno silo.

Prosti pad

Pri prostem padu premaknemo opazovano telo na določeno višino od tal in ga spustimo. Začetna hitrost v navpični smeri je enaka nič, tako da na telo delujeta samo gravitacija in morebitni zračni upor. Pri prostem padu nas običajno zanima čas padanja, torej čas od spusta objekta do trenutka, ko telo prileti na tla, in hitrost v navpični smeri, s katero telo trči ob tla (končna hitrost). Ko uporabnik določi, da bo opazoval prosti pad, lahko s pomočjo drsnika opazovanemu telesu bolj natančno določi začetno višino. Izbira lahko višino na intervalu od nič do 30 metrov s korakom 0,5 metra. Prav tako lahko izbere vmesno višino od tal, na kateri želi zabeležiti hitrost telesa. Aplikacija ob zagonu simulacije zabeleži čas padanja, končno hitrost in morebitno vmesno hitrost na določeni višini, začetno potencialno energijo ter končno kinetično energijo.

Met navzdol

Pri metu navzdol opazovano telo z začetno hitrostjo v navpični smeri, različno od nič, iz dane višine vržemo proti tlom. Poleg gravitacije in morebitnega zračnega upora na telo deluje tudi sila, katere rezultat je nenadna sprememba hitrosti. Ta sprememba hitrosti odraža začetno hitrost meta. Ko uporabnik izbere, da bo telo vrgel navzdol, lahko poleg začetne in vmesne višine z drsnikom nastavlja tudi začetno hitrost meta. Izbere lahko začetno hitrost z intervala od 0 do 100 metrov na sekundo s korakom 0.5 metra na sekundo. V primeru, da uporabnik začetno hitrost nastavi na 0, gibanje preide v prosti pad. Aplikacija ob zagonu simulacije beleži enake spremenljivke kot pri prostem padu.

Navpični met

Pri navpičnem metu opazovano telo z začetno hitrostjo v navpični smeri, različno od nič, iz dane višine vržemo navpično navzgor. Telo lahko vržemo s tal ali z določene višine nad tlemi. Višino in začetno hitrost lahko prav tako kot pri metu navzdol nastavljamo z drsnikoma. Začetni parametri navpičnega meta so pravzaprav enaki kot pri metu navzdol, le da je smer začetne hitrosti ravno nasprotna. V primeru, da začetno hitrost navpičnega meta nastavimo na 0, gibanje prav tako kot pri metu navzdol preide v prosti pad. Vendar pa pri navpičnem metu običajno ne spremljamo enakih parametrov kot pri prostem padu in metu navzdol. Običajno nas pri navpičnem metu zanima največja višina, ki jo telo doseže, in čas od začetka meta do trenutka, ko jo je doseglo. Omenjena parametra imenujemo dvižna višina in dvižni čas. Celoten čas navpičnega meta je enak vsoti dvižnega časa in časa padanja z dvižne višine in je enak dvakratniku dvižnega časa. Velja namreč zakonitost, da je čas dviganja enak času padanja. Aplikacija poleg vseh omenjenih parametrov zabeleži tudi začetno potencialno in končno kinetično energijo.

Vodoravni met

Pri vodoravnem metu prav tako kot pri navpičnem metu ali metu navzdol telo vržemo z začetno hitrostjo, ki je različna od nič. Vektor začetne hitrosti je vzporeden tlom. Tudi pri vodoravnem metu telesu nastavljamo velikost začetne hitrosti in začetno višino. Če je začetna hitrost enaka nič, gibanje preide v prosti pad. Ko telo vržemo vodoravno, je njegov tir gibanja parabola. Pri vodoravnem metu nas običajno zanima čas gibanja od trenutka, ko telo vržemo, do trenutka, ko telo pade na tla in domet. Domet nam pove, kako daleč je pristalo telo, torej predstavlja vodoravno razdaljo med začetno in končno lego telesa.

Potiskanje telesa

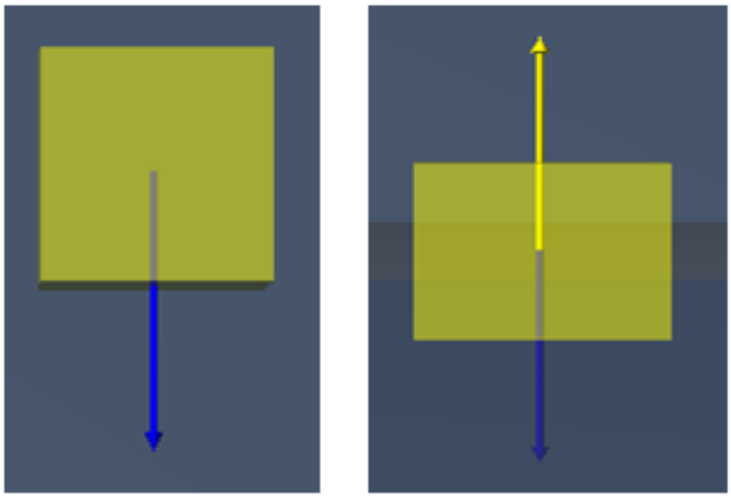
Vsi predhodno opisani meti imajo lastnost, da nenadna sila povzroči spremembo v hitrosti in s tem tudi tir gibanja telesa. Pri potiskanju telesa pa sila na telo ne deluje nenadno, ampak konstantno skozi več časovnih okvirjev. Če želimo telo potiskati, ne nastavljamo hitrosti, ampak konstantno silo, ki bo začela delovati nanj ob zagonu simulacije. Uporabnik lahko z drsnikom spreminja tako velikost vlečne oziroma potisne sile kot tudi njen kot glede na vodoravno podlago (tla). Opazovanje tega gibanja je smiselno le za telesa, ki so na tleh. Opazovanje potiskanja telesa po podlagi je namenjeno demonstraciji delovanja sil na telo, ki drsi po podlagi.

5.2 Sistemi vektorjev

S klikom na ustrezno stikalo je v aplikaciji možno vključiti ali izključiti prikaz vektorjev za krogle, kocke ali kvadre. Uporabnik se lahko odloči za prikaz vektorjev hitrosti in pospeška telesa ter sil na telo. Prijemališče sistema vektorjev, opisanega v nadaljevanju, je zaradi poenostavitve v vseh primerih postavljeno v središče telesa. Izjema je vlečna sila, ki ima prijemališče na površini telesa.

Pri vklopu vektorjev hitrosti ali pospeška se v središče 3D telesa postavi ortogonalni sistem vektorjev ($\mathbf{v_x}, \mathbf{v_y}, \mathbf{v_z}$ ali $\mathbf{a_x}, \mathbf{a_y}, \mathbf{a_z}$). Da uporabniku takoj zagotovimo povratno informacijo, je prikaz ortogonalnega sistema vektorjev ob vklopu simboličen in ni odraz realnega stanja. Šele ob zagonu simulacije se telesa začnejo gibati, posledično pa se velikosti in smeri vektorjev v vseh koordinatnih smereh spreminjajo premo sorazmerno z dejansko hitrostjo oziroma pospeškom. Trenutno velikost in njeno smer lahko v vsakem okvirju odčitamo neposredno iz Unityjevih togih teles, pospešek pa v enakomernih časovnih intervalih računamo po formuli $\mathbf{a} = \mathbf{v}\Delta\mathbf{t}$.

Ob vklopu sil, ki delujejo na telo, se ne glede na to, ali je simulacija v teku ali ne, vedno pojavi vektor sile teže. Smer vektorja je vedno navpično navzdol, njegova velikost pa je premo sorazmerna z dejansko maso objekta. V primeru, da se telo nahaja na vodoravni podlagi, nanj pa ne deluje nobena konstantna sila, aplikacija poleg vektorja sile teže izriše še vektor sile podlage, ki je nasprotno enaka sili teže. Vizualizacijo opisanih primerov prikazuje slika 5.1. Če na telo, ki se nahaja na vodoravni podlagi, delujemo s konstantno vlečno silo, potem aplikacija poleg vektorja vlečne sile izriše še vektor sile teže ter vektor sile podlage, razstavljene na normalno in vzporedno komponento. V primeru, da se telo nahaja na klancu, se sila teže razstavi na dinamično in statično komponento sile teže, katerih velikost je enaka $\mathbf{F_g} \sin\alpha$ ter $\mathbf{F_g} \cos\alpha$. Dinamična komponenta je vzporedna klancu, statična pa je pravokotna nanj. Če povečamo naklon klanca, se ustrezno poveča dinamična in zmanjša statična komponenta in obratno. Pri silah na klancu običajno razstavimo tudi silo podlage, in sicer na normalno ter vzdolžno komponento. Normalna komponenta sile podlage je nasprotno enaka statični komponenti sile teže, vzdolžna komponenta sile podlage pa je nasprotno enaka dinamični komponenti sile teže. Če telo miruje na klancu ali se po njem giblje enakomerno, potem je v ravnovesju in rezultanta vseh sil znaša nič. Razstavitev sile podlage in vizualizacijo vlečne sile ter sile na klancu prikazuje slika 5.2.

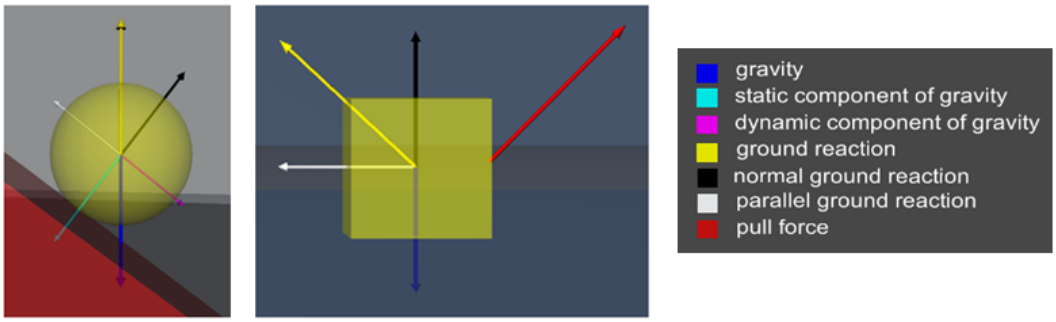


Slika 5.1: Vektor sile teže na telo, ki pada, in vektorja sile teže ter podlage na telo, ki miruje na vodoravni podlagi.

V aplikaciji lahko telesom nastavljamo tudi zračni upor, ki ustrezno upočasnjuje morebitno padanje telesa, vendar se za vizualizacijo sile zračnega upora z vektorjem nismo odločili.

5.2.1 Prosojnost

Uporabnik lahko vključi ali izključi delno prosojnost materialov, iz katerih so zgrajeni 3D objekti v sceni. Delno prosojnost omogoča uporaba barvnega modela RGBA, ki poleg kanala za rdečo, zeleno in modro barvo omogoča kanal za določanje stopnje prosojnosti. Ko je kanal za prosojnost, t.i. alfa kanal, nastavljen na maksimalno vrednost, je material neprosojen, ko pa ima minimalno vrednost, je popolnoma prosojen [24]. Možnost vklopa delne prosojnosti je smiselna v primeru, ko uporabnik vključi prikaz vektorjev, kajti velikost vektorjev je lahko manjša od samega telesa. Pri postavitvi sistema vektorjev v središče telesa so v opisanem primeru vektorji nevidni. Z vklopom prosojnosti se poleg 3D objektov, ki jih lahko uporabnik spreminja, nastavi tudi prosojnost sten in tal. Na ta način zopet zagotovimo, da upo-



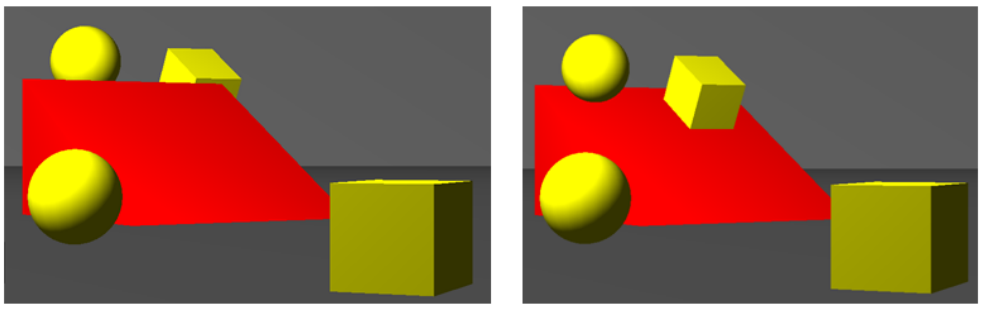
Slika 5.2: Vizualizacija sil na klancu in sil pri vlečenju telesa. V aplikaciji se ob vklopu prikaza vektorjev sil za lažje razločevanje pojavi tudi legenda.

rabnik lahko vidi vektorje npr. ob trku telesa v steno ali tla, saj bi bili v nasprotnem primeru vsaj deloma nevidni.

Delna prosojnost materialov, iz katerih so objekti na sceni, je znotraj razvoja aplikacije povzročila nekaj težav pri njihovem upodabljanju. Po navedbah uradne Unityjeve dokumentacije je uporaba prosojnih senčilnikov znan povzročitelj nepravilnosti pri upodabljanju, predvsem takrat, ko je na sceni veliko prekrivajočih se objektov s prosojnimi senčilniki [25]. Problem smo rešili z uporabo odprtokodnega senčilnika, ki ni vgrajen v Unity in omogoča globinsko sortiranje delno prosojnih objektov [26]. Stanje pred in po uporabi omenjenega senčilnika prikazuje slika 5.3.

5.3 Simulacija

Dokler ne poženemo simulacije, 3D objekti ne bodo odražali delovanja realnih sil. Na ta način uporabniku zagotovimo nemoteno spreminjanje sistema, nastavljanje začetnih parametrov posameznih teles in premikanje po prostoru, ne da bi telesa nemudoma padla na tla zaradi delovanja gravitacije. Delovanje sil, tako gravitacije kot tudi morebitnih vlečnih ali nenadnih sil, se vklopi šele ob zagonu simulacije. Simulacijo poženemo s klikom na zeleni



Slika 5.3: Prikaz upodobitve iste postavitve objektov z dvema različnima senčilnikoma. Levo je prikazana upodobitev z enim izmed Unityjevih vgrajenih senčilnikov, desno je prikazana pravilna upodobitev z odprtokodnim senčilnikom, ki je uporabljen v aplikaciji.

gumb ali z zamahom roke navzdol, ustavimo pa s klikom na rdeči gumb ali zamahom roke navzgor. Ob ustavitvi simulacije objekti zamrznejo, tako da Unityjev fizikalni pogon z njimi ne upravlja več [27]. Alternativna rešitev bi bila zamrznitev položaja in rotacije teles, vendar s stališča izvajanja to ni optimalno, saj bi ti objekti še vedno obremenjevali procesor.

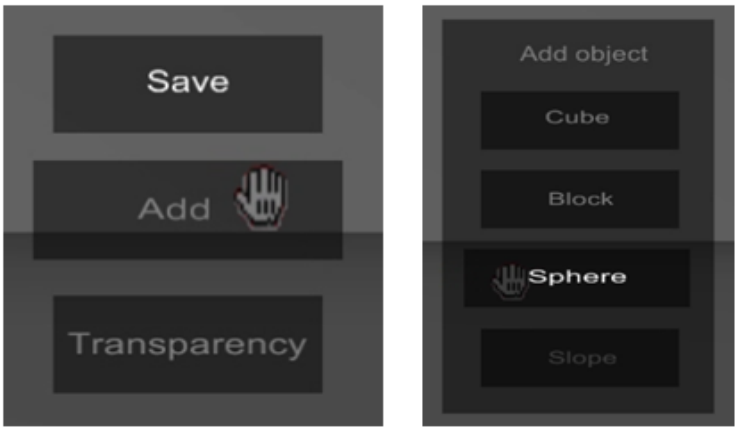
Poglavje 6

Primeri uporabe aplikacije

Za ilustracijo delovanja aplikacije si oglejmo nekaj tipičnih primerov uporabe. Predpostavimo, da je cilj uporabe aplikacije rešitev danega fizikalnega problema. Problemi naj bodo podani v obliki besedilnih nalog. Spodnji primeri opisujejo kratke opise postopkov, ki uporabniku omogočajo osnovno interakcijo z aplikacijo ter pot do rešitve zastavljenega fizikalnega problema.

Primer 1 - prosti pad
Telo spustimo na tla z višine 30 m. Koliko časa preteče, preden telo prileti na tla? Kdaj telo doseže višino 22.5 metrov in kakšna je takrat njegova hitrost?

V vstopnem meniju, prikazanem na sliki 4.3, izberemo možnost *New scene*, kar bo ustvarilo prazno sceno. Na sceno s klikom na gumb *Add* dodamo nov objekt, ki naj bo krogla. Postavimo jo na poljubno mesto in jo odložimo tako, da roko iztegnemo v smeri naravnost proti senzorju. Opisano interakcijo z uporabniškimi kontrolami prikazuje slika 6.1. Pritisnemo na novo dodano kroglo in v meniju, ki se odpre, kliknemo na gumb *Physics*. Izberemo možnost *Free fall*, saj na ta način omogočimo podrobnejše nastavljanje parametrov in izpis vrednosti opazovanih parametrov v oznako z rezultati. Ponovno pritisnemo kroglo in zamahnemo v desno, tako da se nam odpre meni za manipulacijo parametrov. Popravimo višino krogle tako, da bo ustrezala navodilom naloge. Prav tako nastavimo vmesno višino, pri kateri želimo za-



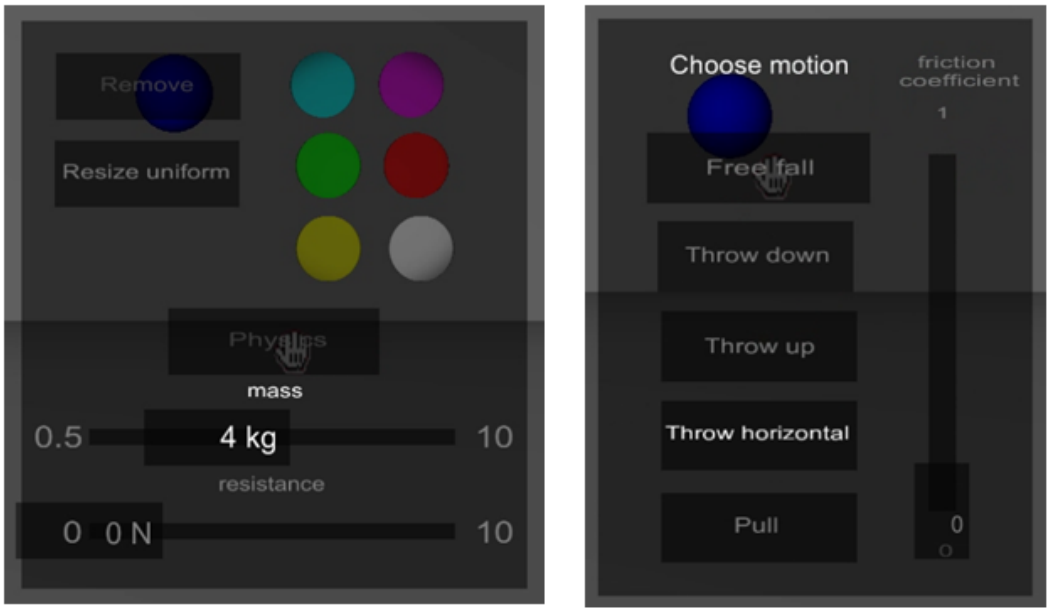
Slika 6.1: Dodajanje novega objekta na sceno in izbor tipa objekta, ki bo na novo dodan v sceno.

beležiti hitrost krogle. Z zamahom v levo zapremo meni, nato pa z zamahom navzdol ali klikom na zeleni gumb poženemo simulacijo.

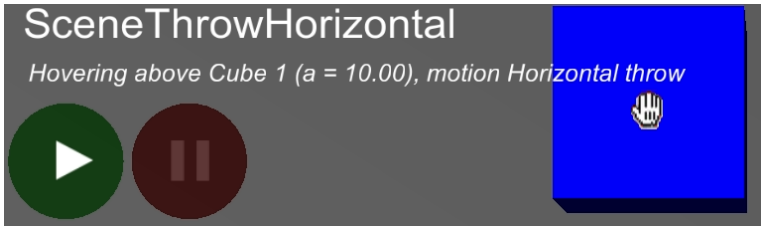
Primer 2 - vodoravni met

Kocko vodoravno vržemo z višine 20 m z začetno hitrostjo 25 m/s. Kakšna je lega krogle po 1 sekundi gibanja? Kakšen je domet? Pod katerim kotom telo prileti na tla in kakšna je skupna končna hitrost ob trku s tlemi?

V vstopnem meniju iz nabora ponujenih scen izberemo sceno *Horizontal throw*. Ustvari se sistem s kocko, ki ji je že dodana skripta za spremljanje fizikalnega obnašanja vodoravnega meta. To lahko vidimo tako, da kurzor postavimo nad kocko. Slika 6.3 prikazuje izpis informacije o gibanju kocke, ki se bo izvedlo ob zagonu simulacije in rob kocke. Ob pritisku na kocko se odpre meni za osnovno manipulacijo telesa. Kot smo že omenili v prejšnjem primeru, lahko z zamahom roke v desno pridemo do menija, kjer nastavljamo začetne parametre fizikalnega gibanja. Vrednosti teh parametrov so sprva privzete, zato jih s pomočjo drsnikov nastavimo tako, da bodo skladni s podatki iz besedila naloge. Ko poženemo simulacijo, se začne izvajati gibanje z izbranimi začetnimi parametri. Vodoravni met spremljamo od zagona si-



Slika 6.2: Osnovni meni za manipulacijo telesa in meni za dodajanje opazovanega fizikalnega gibanja.



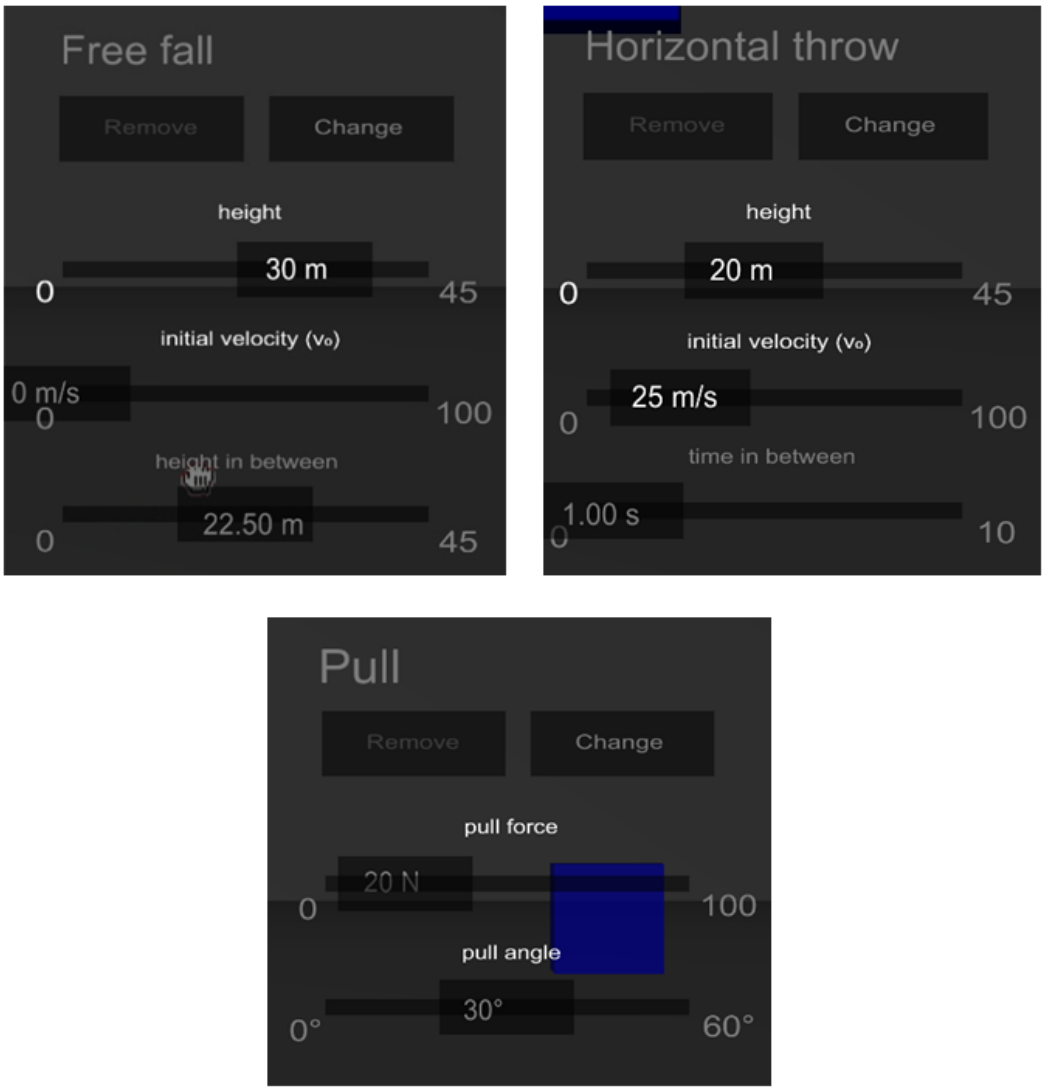
Slika 6.3: Lebdenje kurzorja nad fizikalnim objektom in povratne informacije.

mulacije do trka telesa s tlemi. Aplikacija ob koncu gibanja v oznako izpiše vse bistvene parametre. Rešitve problemov za vse primere, navedene v tem poglavju, prikazuje slika 6.5. Na tej točki nam je aplikacija izračunala rešitev problema in podala odgovor na vsa zastavljena vprašanja.

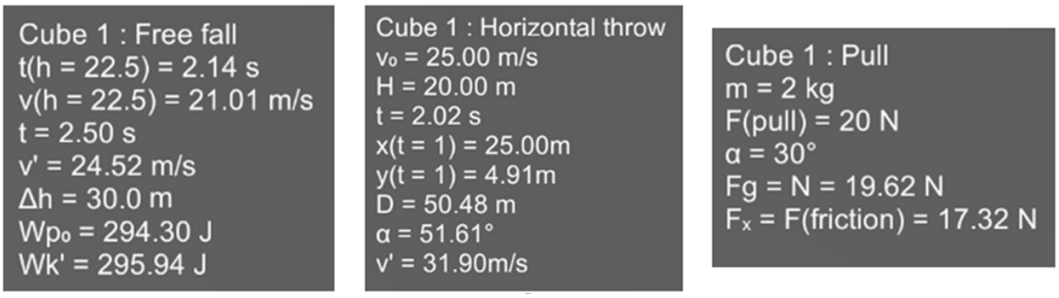
Primer 3 - vlečenje klade

Klado z maso 2 kg s konstantno silo vlečemo po vodoravni podlagi. Izračunaj silo trenja, če veš, da klado vlečemo s silo 20 N pod kotom 30° glede na vodoravnico.

Na enega od zgoraj opisanih načinov (bodisi z izbiro predpripravljene scene za vlečenje klade, bodisi z dodajanjem fizikalnega gibanja izbranemu objektu) lahko dosežemo vlečenje objekta po vodoravni podlagi. V skladu z navodili naloge iz menija za manipulacijo fizikalnih parametrov podobno kot v prejšnjih primerih z drsniki določimo velikost in kot vlečne sile. Ob zagonu simulacije aplikacija izračuna silo trenja, ta je zaradi enakomernega gibanja, ki je posledica vlečenja klade s konstantno silo, nasprotno enaka vzdolžni komponenti vlečne sile. Vlečna sila vizualno ni razstavljena na svojo vzdolžno in navpično komponento tako kot sila podlage, vendar aplikacija izračuna njeno velikost, saj je v tem primeru po velikosti enaka sili trenja.



Slika 6.4: Meniji za nastavljanje fizikalnih parametrov s pomočjo drsnikov. Parametri so nastavljeni v skladu z vrednostmi iz besedilnih nalog. Iz menijev je razvidno, katere parametre je mogoče nastavljati za opazovana gibanja, ki so opisana v zgornjih treh primerih.



Slika 6.5: Rešitve opisanih problemov kot jih prikaže aplikacija za gibanja, ki ustrezajo začetnim parametrom opisanih primerov fizikalnih nalog.

Poglavje 7

Sklep

Čeprav smo prepričani, da ima razvita aplikacija potencial za aktivno uporabo pri poučevanju, se zavedamo njenih pomanjkljivosti. Za realno oceno uporabne vrednosti aplikacije na področju izobraževanja bi bila potrebna evalvacija s strani učiteljev fizike, ki v okviru diplomskega dela ni bila izvedena. V nadaljevanju bomo analizirali bistvene vidike, ki so po našem mnenju kritični za uvedbo naše aplikacije kot učnega pripomočka.

Prvi korak k uporabi razvite aplikacije v učilnicah je utrezna strojna oprema. Predvidevamo, da sta v večini moderno opremljenih učilnic računalnik, ki je dovolj zmogljiv za tekoče izvajanje aplikacije, in projektor. V tem primeru je potrebno kupiti le Kinect. V primerjavi s stroški nakupa katerega drugega interaktivnega pripomočka za učilnice, npr. interaktivne table, je Kinect razmeroma poceni.
V nadaljevanju bomo z izrazom uporabnik aplikacije označili tistega, ki krmili aplikacijo preko naravnega uporabniškega vmesnika. Uporabnike aplikacije, ki so v primeru morebitne uporabe v učilnicah učitelji, bi bilo najprej potrebno naučiti njene uporabe s pomočjo ročnih kretenj. Omeniti je potrebno, da naravni uporabniški vmesnik razvite aplikacije predstavlja hkrati njeno prednost in pomanjkljivost. Prednost njegove uporabe vidimo predvsem v dejstvu, da bi z njim učitelji utegnili povečati zanimanje in motivacijo

pri učencih. Menimo, da je to pomemben dosežek pri učenju tematik, ki se učencem lahko zdijo nezanimive. Pomanjkljivosti uporabe naravnega uporabniškega vmesnika pa vključujejo kompromis med interaktivnostjo in fizikalno natančnostjo. Pri postavljanju lastnih fizikalnih scen je namreč težko posnemati stanje kompleksnejših sistemov. Aplikacija ponuja zgolj nekaj elementov, iz katerih lahko sestavljamo sisteme, vendar se že pri postavljanju nove kocke ali krogle na klanec jasno vidi pomanjkanje natančnosti. Aplikacija v trenutni obliki namreč ne omogoča rotacije teles. Ta problem lahko zaobidemo tako, da v aplikaciji ponudimo nabor vnaprej pripravljenih scen, ki se v fizikalnih nalogah pogosto pojavljajo, uporabnik pa se izogne opisani pomanjkljivosti. Poleg tega zaradi ročne navigacije po aplikaciji uporabnik porabi več časa za nastavljanje parametrov ali postavljanje fizikalnega sistema. Vnašanje vrednosti parametrov preko naravnega uporabniškega vmesnika je manj natančno kot vnašanje vrednosti s pomočjo miške in tipkovnice.

Omenimo še to, da bi bilo aplikacijo mogoče vsebinsko nadgraditi tako, da bi pokrivala še druga področja fizike, ali pa vsaj dodatne tipe nalog za trenutno pokrita področja. Na ta način bi lahko aplikacijo razširili še s čelnimi trki in spremljanjem navora ter gibalne in vrtilne količine, več pozornosti bi lahko posvetili tudi sili zračnega upora in trenju. Nabor elementov, ki jih je mogoče dodajati v fizikalni sistem, bi lahko obogatili z matematičnimi in fizikalnimi nihali, škripci in vzvodi. Čeprav je aplikacija usmerjena v poučevanje fizike, je bil precejšnji del razvoja aplikacije usmerjen v interakcijo preko naravnega uporabniškega vmesnika in razvoj interaktivnih uporabniških kontrol. Ne moremo torej trditi, da je fizikalni vidik aplikacije njena najbolj dodelana komponenta. Kljub temu smo pokazali, da je možnosti za izboljšave in nadgradnje fizikalnega vidika aplikacije veliko. Prepričani smo, da je prihodnost uporabe aplikacije v izobraževalne namene odvisna prav od spektra fizikalnih vsebin, ki jih bo sposobna ponuditi.

Literatura

[1] Kinect for Windows Sensor Components and Specifications. [Online]. Dostopno na: http://msdn.microsoft.com/en-us/library/jj131033.aspx. [Dostopano: 5.7.2014].

[2] Wikipedia - Kinect. [Online]. Dostopno na: http://en.wikipedia.org/wiki/Kinect. [Dostopano: 5.7.2014].

[3] A. Da Gama, T. Chaves, L. Figueiredo, V. Teichrieb, "Improving motor rehabilitation process through a natural interaction based system using kinect sensor", *3D User Interfaces, 2012 IEEE Symposium*, str. 145-146, 2012

[4] Hui-Mei Justina Hsu, "The Potential of Kinect in Education", *International Journal of Information and Education Technology*, št. 1, zv. 5, str. 365-370, 2011

[5] KinectEDucation. [Online]. Dostopno na: http://www.kinecteducation.com. [Dostopano: 7.7.2014].

[6] Wikipedia - Unity (game engine). [Online]. Dostopno na: http://en.wikipedia.org/wiki/Unity_(game_engine).[Dostopano: 20.7.2014].

[7] Unity - Manual: The GameObject-Component Relationship. [Online]. Dostopno na: http://docs.unity3d.com/Manual/TheGameObject-ComponentRelationship.html. [Dostopano: 16.8.2014].

[8] Unity - Scripting API: MonoBehaviour. [Online]. Dostopno na: http://docs.unity3d.com/ScriptReference/MonoBehaviour.html. [Dostopano: 16.8.2014].

[9] Unity - Manual: Colliders. [Online]. Dostopno na: http://docs.unity3d.com/Manual/CollidersOverview.html. [Dostopano: 16.8.2014].

[10] Asset Store - Kinect with MS-SDK. [Online]. Dostopno na: https://www.assetstore.unity3d.com/en/#!/content/7747. [Dostopano: 6.6.2014].

[11] Asset Store - KinectExtras with MsSDK. [Online]. Dostopno na: https://www.assetstore.unity3d.com/en/#!/content/10492. [Dostopano: 6.6.2014].

[12] Wikipedia - Blender (software). [Online]. Dostopno na: http://en.wikipedia.org/wiki/Blender_(software). [Dostopano: 25.7.2014].

[13] J. Bishop, *C# 3.0 Design Patterns*, O'Reilly Media, 2007, pogl. 5

[14] J. Albahari, B. Albahari, *C# 4.0 in a nutshell, 4th Edition*, O'Reilly Media, 2010, pogl. 16

[15] Wikipedia - Serialization. [Online]. Dostopno na: http://en.wikipedia.org/wiki/Serialization. [Dostopano: 18.7.2014].

[16] Unity - Manual: Publishing builds. [Online]. Dostopno na: http://docs.unity3d.com/Manual/PublishingBuilds.html. [Dostopano: 25.7.2014].

[17] Wikipedia - Ray casting. [Online]. Dostopno na: http://en.wikipedia.org/wiki/Ray_casting. [Dostopano: 20.7.2014].

[18] A. Boreskov, E. Shikin, *Computer Graphics: From Pixels to Programmable Graphics Hardware*, CRC Press, 2013, pogl. 8

[19] Unity - Scripting API: Physics Raycast. [Online]. Dostopno na: http://docs.unity3d.com/ScriptReference/Physics.Raycast.html. [Dostopano: 20.7.2014].

[20] Unity - Manual: Layers. [Online]. Dostopno na: http://docs.unity3d.com/Manual/Layers.html. [Dostopano: 20.7.2014].

[21] Wikipedia - Physics Engine. [Online]. Dostopno na: http://en.wikipedia.org/wiki/Physics_engine. [Dostopano: 3.8.2014].

[22] NVIDIA PhysX FAQ. [Online]. Dostopno na: http://www.nvidia.com/object/physx_faq.html. [Dostopano: 3.8.2014].

[23] Unity - Manual: Rigidbody. [Online]. Dostopno na: http://docs.unity3d.com/Manual/class-Rigidbody.html. [Dostopano: 4.8.2014].

[24] Wikipedia - RGBA Color Space. [Online]. Dostopno na: http://en.wikipedia.org/wiki/RGBA_color_space. [Dostopano: 4.8.2014].

[25] Unity - Manual: Transparent Diffuse. [Online]. Dostopno na: http://docs.unity3d.com/Manual/shader-TransDiffuse.html. [Dostopano: 4.8.2014].

[26] Transparent Alpha Vertex-Lit With Z. [Online]. Dostopno na: http://wiki.unity3d.com/index.php/Transparent_Alpha_Vertex_Lit_With_Z. [Dostopano: 4.8.2014].

[27] Unity - Manual: Rigidbodies Overview. [Online]. Dostopno na: http://docs.unity3d.com/Manual/RigidbodiesOverview.html. [Dostopano: 3.8.2014].